Release Notes and Installation Guide for

The Booch Components™

Sampler

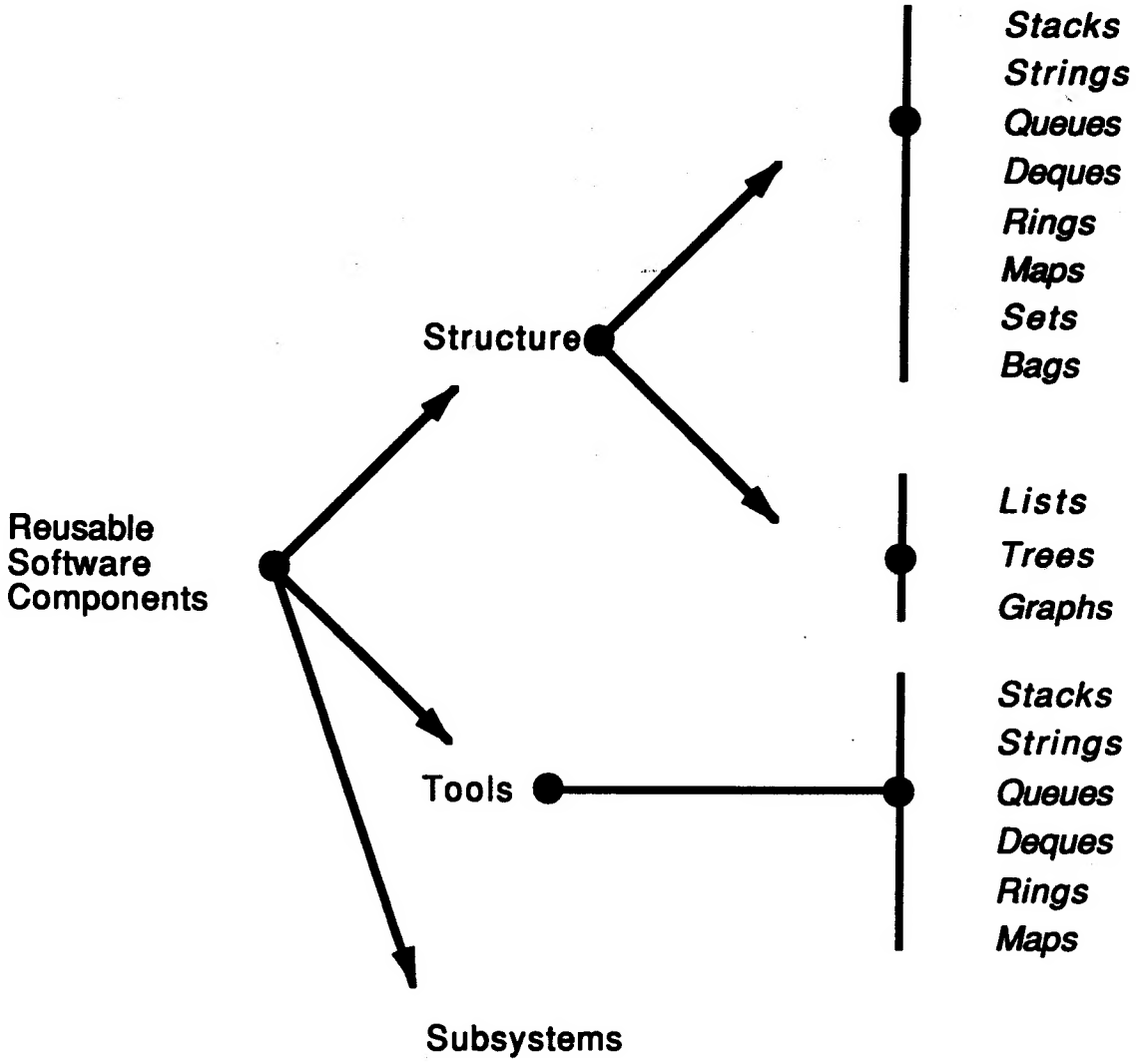
Meridian Software Systems

October 1990

Description

This document provides a brief overview of The Booch Components Sampler, a set of five components selected from The Booch Components, which is a collection of over 500 highly reusable Ada components implementing data structures such as lists, stacks, and queues. Definitive coverage of the rationale and design philosophy behind the components is presented in the text *Software Components with Ada* (Addison-Wesley, 1988) by Grady Booch, one of the best-known exponents of software engineering with Ada.

In that work, Booch presents a categorization of reusable components which is depicted in the figure below. The Booch components include the collection of structures listed above. In Ada terms, these components rely heavily upon the use of generics, and provide the greatest degree of flexibility and tailorability for the user.


Reusable
Software
Components
Structure
Tools
Subsystems
Stacks
Strings
Queues
Deques
Rings
Maps
Sets
Bags
Lists
Trees
Graphs
Stacks
Strings
Queues
Deques
Rings
Maps

The Booch Components Sampler is intended to provide the user with an introductory familiarity with the entire component set. Five components have been selected, including packages for Stack, List, Set, and Tree abstract data types, and a utility package, Sort. The components have been selected for their ease of application, and high utility.

In terms of Booch's categories of forms, the structures provided all share the following characteristics: they are sequential (not tasking oriented), bounded (the size of the object is bounded), non-iterator (no iterator functions are provided). Specifications of these components are provided below, and are included on-line as well in the ada\sbooch directory.

The five components are pre-compiled into the Ada library, booch.lib, which is installed in the ada\sbooch directory.

The components can be used by linking to the Sampler library. For instructions on linking libraries, please refer to the Meridian Ada Compiler User's Guide.

Component	Full Name
Stack	Stack_Sequential_Bounded_Managed_Noniterator

The Stack component provides a simple implementation of a generic stack. To use the component, link to the Sampler library, and import the component by use of a 'with' clause.

Since the component is a generic, it must be instantiated prior to use. Select the type of information you wish to manipulate on a stack, and pass that as a generic argument. (For a full description of generics and their use, refer to the Reference Manual for the Ada Programming Language, Chapter 12. Generics).

The basic notion is to place (push) and remove (pop) items to and from the stack. The other operations provided allow more efficient use of the abstraction. The underlying implementation relies on array manipulation, but since the Stack type exported by the package is limited private, implementation details are hidden.

```
generic
    type Item is private;
package Stack_Sequential_Bounded_Managed_Noniterator is

    type Stack(The_Size : Positive) is limited private;

    procedure Copy   (From_The_Stack : in     Stack;
                      To_The_Stack   : in out Stack);
    procedure Clear  (The_Stack      : in out Stack);
    procedure Push   (The_Item       : in     Item;
                      On_The_Stack   : in out Stack);
    procedure Pop    (The_Stack      : in out Stack);

    function Is_Equal (Left      : in Stack;
                       Right     : in Stack) return Boolean;
    function Depth_Of (The_Stack : in Stack) return Natural;
    function Is_Empty (The_Stack : in Stack) return Boolean;
    function Top_Of   (The_Stack : in Stack) return Item;

    Overflow   : exception;
    Underflow  : exception;

private
    ...
end Stack_Sequential_Bounded_Managed_Noniterator;
```

Component	Full Name
List	List_Single_Bounded_Managed

The List component provides a bounded length list, with a simple suite of list operations. Like the stack example, it must be instantiated to be used, but the size of the list must be fixed at the point of instantiation.

The construct operation provides the ability to add elements to the head of a list, while the clear operation wipes out the contents of a list. Copying from one list to another is supported by the copy operation.

Set_Head modifies the value of the item at the head of the list, while Set_Tail replaces the tail of the list (everything but the head) with another list.

```
generic
    type Item is private;
    The_Size : in Positive;
package List_Single_Bounded_Managed is

    type List is private;

    Null_List : constant List;

    procedure Copy      (From_The_List : in     List;
                         To_The_List   : in out List);
    procedure Clear     (The_List      : in out List);
    procedure Construct (The_Item      : in     Item;
                         And_The_List  : in out List);
    procedure Set_Head  (Of_The_List   : in out List;
                         To_The_Item   : in     Item);
    procedure Swap_Tail (Of_The_List   : in out List;
                         And_The_List  : in out List);

    function Is_Equal   (Left      : in List;
                         Right     : in List) return Boolean;
    function Length_Of  (The_List  : in List) return Natural;
    function Is_Null    (The_List  : in List) return Boolean;
    function Head_Of    (The_List  : in List) return Item;
    function Tail_Of    (The_List  : in List) return List;

    Overflow     : exception;
    List_Is_Null : exception;

private
    ...
end List_Single_Bounded_Managed;
```

Component	Full Name
Set	Set_Simple_Sequential_Bounded_Managed_Noniterator

The set abstraction is a means of implementing set theory in Ada. The component is generic, and must be instantiated over the desired type. The size of a set is provided by the discriminant for the Set type at the point of declaration.

The clear, copy, add, and remove operations serve the obvious purpose indicated by their names.

The remaining operations correspond to the equivalent set theoretic functions (e.g., intersection).

```
generic
    type Item is private;
package Set_Simple_Sequential_Bounded_Managed_Noniterator is

    type Set(The_Size : Positive) is limited private;

    procedure Copy          (From_The_Set : in     Set;
                             To_The_Set   : in out Set);
    procedure Clear         (The_Set      : in out Set);
    procedure Add           (The_Item     : in     Item;
                             To_The_Set   : in out Set);
    procedure Remove        (The_Item     : in     Item;
                             From_The_Set : in out Set);
    procedure Union         (Of_The_Set   : in     Set;
                             And_The_Set  : in     Set;
                             To_The_Set   : in out Set);
    procedure Intersection  (Of_The_Set   : in     Set;
                             And_The_Set  : in     Set;
                             To_The_Set   : in out Set);
    procedure Difference    (Of_The_Set   : in     Set;
                             And_The_Set  : in     Set;
                             To_The_Set   : in out Set);

    function Is_Equal            (Left       : in Set;
                                  Right      : in Set) return Boolean;
    function Extent_Of           (The_Set    : in Set) return Natural;
    function Is_Empty            (The_Set    : in Set) return Boolean;
    function Is_A_Member         (The_Item   : in Item;
                                  Of_The_Set : in Set)  return Boolean;
    function Is_A_Subset         (Left       : in Set;
                                  Right      : in Set)  return Boolean;
    function Is_A_Proper_Subset  (Left       : in Set;
                                  Right      : in Set)  return Boolean;

    Overflow           : exception;
    Item_Is_In_Set     : exception;
    Item_Is_Not_In_Set : exception;

private
```

```
   ...
end Set_Simple_Sequential_Bounded_Managed_Noniterator;
```

Component	Full Name
Tree	Tree_Binary_Single_Bounded_Managed

The tree component implements a binary tree, whose size and type are fixed by generic instantiation.

The procedures implement fundamental operations, with copy, clear, and construct working in the obvious way. Set_Item changes the value of the root of the tree to the provided value. Swap_Child exchanges the selected child of a tree with the entire tree provided.

```
generic
    type Item is private;
    The_Size : in Positive;
package Tree_Binary_Single_Bounded_Managed is

    type Tree is private;

    type Child is (Left, Right);

    Null_Tree : constant Tree;

    procedure Copy       (From_The_Tree : in     Tree;
                          To_The_Tree   : in out Tree);
    procedure Clear      (The_Tree      : in out Tree);
    procedure Construct  (The_Item      : in     Item;
                          And_The_Tree  : in out Tree;
                          On_The_Child  : in     Child);
    procedure Set_Item   (Of_The_Tree   : in out Tree;
                          To_The_Item   : in     Item);
    procedure Swap_Child (The_Child     : in     Child;
                          Of_The_Tree   : in out Tree;
                          And_The_Tree  : in out Tree);

    function Is_Equal (Left      : in Tree;
                       Right     : in Tree) return Boolean;
    function Is_Null  (The_Tree  : in Tree) return Boolean;
    function Item_Of  (The_Tree  : in Tree) return Item;
    function Child_Of (The_Tree  : in Tree;
                       The_Child : in Child) return Tree;

    Overflow     : exception;
    Tree_Is_Null : exception;

private
    ...
end Tree_Binary_Single_Bounded_Managed;
```

Component	Full Name
Sort	Quicksort

The quicksort component offers a high performance sort utility in generic form. The type, and range of the array to be sorted must be provided as arguments to the generic instantiation, along with a "less than" operation.

```
generic
    type Item  is private;
    type Index is (<>);
    type Items is array(Index range <>) of Item;
    with function "<" (Left  : in Item;
                       Right : in Item) return Boolean;
package Quick_Sort is

    procedure Sort (The_Items : in out Items);

end Quick_Sort;
```

™ The Booch Components is a trademark of Wizard Software, Inc.